REMARKS

Claims 59-150 are now pending. No claims are allowed. Claims 1-58 were rejected in an office action dated January 23, 2003. Claims 1-3, 9-11, 13, 15-16, 22-24, and 26 were rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which the Applicant regards as the invention.[1] Claims 1, 3-5, 9-11, 13-15, 16-18, 22-24, 26-29, 31, 32, 36-40, 42, 43, 45-48, 52-55, and 58 were rejected under 35 U.S.C. § 102(e) as being anticipated by Wilkinson et al.[2] [3] Claims 12, 25, and 56-57 were rejected under 35 U.S.C. § 103(a) as obvious over Wilkinson et al.[4] Claims 2, 6-8, 19-21, 30, 33-55, 41, 44, and 49-51 were rejected under 35 U.S.C. § 103(a) as obvious over Wilkinson et al. in view of Benaloh et al.[5] [6] With this Amendment, original claims 1-58 have been cancelled without prejudice or disclaimer. No "new matter" has been added by the Amendment.

The Applicants submit new claims 59-150 are distinguished over the cited references for the reasons set forth below. A summary of the new claims follows a discussion of the present invention and cited references.

The Applicants assert that the cited references set forth a problem, but not the solution provided in the Applicants' new claims.

---

[1] Office Action dated January 23, 2003, ¶ 3.
[2] USP 6,308,317.
[3] Office Action ¶ 5.
[4] Office Action ¶ 7.
[5] USP 5,724,279.

A problem, as set forth in the Wilkinson et al. reference, is that use of a high level programming language with a smart card or microcontroller is made difficult by the resource-constrained nature of such devices. Wilkinson et al. discloses:

> Microcontrollers differ from microprocessors in many ways. For example, a microprocessor typically has a central processing unit that requires certain external components (e.g., memory, input controls and output controls) to function properly. A typical microprocessor can access from a megabyte to a gigabyte of memory, and is capable of processing 16, 32, or 64 bits of information or more with a single instruction. In contrast to the microprocessor, a microcontroller includes a central processing unit, memory and other functional elements, all on a single semiconductor substrate, or integrated circuit (e.g., a "chip"). As compared to the relatively large external memory accessed by the microprocessor, the typical microcontroller accesses a much smaller memory. A typical microcontroller can access one to sixty-four kilobytes of built-in memory, with sixteen kilobytes being very common.[7]

A solution proposed in Wilkinson et al. is to compact a Java class file into an optimized version by mapping all the strings found in the class file constant pool into integers.[8] A second aspect of Wilkinson et al. reference focuses on renumbering byte codes for optimizing type tests and thus reducing the program space required by the Card JVM.[9]

In contrast, embodiments of the Applicants' invention are directed to a different aspect of the problem, and provide different solutions. More specifically, in one aspect of the Applicant's claimed invention, an application software program comprises a sequence of instructions that were previously converted from at least one class file, said conversion transforming at least one reference to a constant pool to inline data in the instructions. This aspect is completely missing from the cited references.

---

[6] Office Action ¶ 8.
[7] Wilkinson et al. at col. 1 lines 16-19, col. 1 line 62 – col. 2 line 10.
[8] Wilkinson et al. at col. 9 lines 18-41.

The Examiner contends the Benaloh et al. reference discloses the reduction to in-lined instructions.[10] In support of the contention, the Examiner cites the following:

> The three stage computer-implemented method described above with reference to FIGS. 1-7 is used extensively in computing this modular exponentiation. The method significantly improves the speed of computing modular reduction operations, and thus, improves the overall cryptographic process.
>
> This method is also advantageous for use in processors of limited size or space constraint, such as common 8-bit or 16-bit CPUs used on smart cards. The *method* can be "in-lined" to keep intermediate results small without substantially increasing computational costs. Computations involving large integers are partially reduced each time to ensure that the result has a certain size that can be efficiently handled by the processor. The additional "in-line" computation is not very extensive or time consuming, and thus, is not computationally expensive.[11]

Thus, the Benaloh et al. reference discloses inlining a *method* itself, not inlining data in the instructions. The Benaloh et al. reference does not disclose an application software program comprising a sequence of instructions that were previously converted from at least one class file, said conversion transforming at least one reference to a constant pool to inline data *in the instructions*. The Examiner is reminded that the mere absence from a reference of an explicit requirement of a claim cannot be reasonably construed as an affirmative statement that the requirement is in the reference.[12]

Independent claims 59, 77, 95, and 123 contain this distinction and are thus both novel and unobvious over the cited references. Independent claim 59 claims an application software program comprising an object-oriented, verifiable, type-safe and pointer-safe sequence of instructions residing on a computer-readable medium, said instructions comprising operation

---

[9] Wilkinson et al. at col. 11 lines 4-23.
[10] Office Action p. 11.
[11] Wilkinson et al. at col. 18 line 60 to col. 19 line 8. (emphasis added)

codes and operands, said program operable to be loaded to and executed by a resource-constrained device, said instructions previously converted from at least one class file, said conversion transforming at least one reference to a constant pool to inline data in said instructions. Independent claim 77 claims a resource-constrained device configured to execute the application software program. Independent claim 95 claims a method for using the application software program. Independent claim 123 is a means-plus-function claim corresponding to claim 95.

Support for new claims 59, 77, 95, and 123 is provided in the original specification at page 6 lines 14-30, page 14 line 33 to page 15 line 1, page 16 line 13 to page 20 line 7, and FIGS. 5, 6A, 6B, 7A, 7B, 8A and 8B. Support for new claims 59, 77, 95, and 123 is also provided in original claims 6, 19, 33, 49, and 58.

Dependent claims 60-68, 78-86, 96-108, and 124-136 depend from claims 59, 77, 95, and 123, respectively. The base claims being allowable, the dependent claims must also be allowable.

Additionally, the new dependent claims 60-68, 78-86, 96-108, and 124-136 further define the invention and distinguish the prior art.

Dependent claim 60 specifies that one or more of said references to the constant pool are transformed into inline data in operands in one or more of said instructions. Dependent claims 79, 97, and 125 also include this distinction. This is not disclosed in the cited references, and is

[12] *In re Evanega*, 829 F.2d 1110, 4 USPQ2d 1249 (Fed. Cir. 1987).

described in original claims 7, 20, and 34, and in the original specification at page 14 line 33 to page 15 line 1, page 16 line 13 to page 17 line 20, and FIGS. 5, 6A, 6B, 7A, and 7B.

The Examiner contends the Wilkinson et al. reference discloses that some references to the constant pool are transformed into operands in instructions.[13] In support of the contention, the Examiner cites the following:

> In the third pass 64, the card class file converter rebuilds constant references via elimination of the strings used to denote these constants. FIG. 8 shows an example wherein the byte code LDC 80 referring to constant "18" found via an index in the Java class file 24a constant pool 42 may be translated into BIPUSH byte code 82. In this pass the card class file converter 26 modifies the operands to all the byte codes that refer to entries in the Java class file constant pool 42 to reflect their new location in the card class file constant pool 47. FIG. 9 shows an example wherein the argument to a byte code, INVOKESTATIC 90, refers to an entry in the Java class file constant pool 42 that is *modified to reflect the new location of that entry in the card class file constant pool* 47.[14]

Thus, the Wilkinson et al. reference discloses updating operands to byte codes that reference entries in a first constant pool to reflect their new location in a second constant pool. The modified byte codes disclosed by Wilkinson et al. still reference a constant pool, whereas claim 60 specifies that one or more of references to the constant pool are transformed into inline data in operands in one or more of said instructions, obviating the need to refer to the constant pool when executing the instruction. This is not disclosed by the cited references.

Dependent claim 61 specifies that one or more of said references to said constant pool are transformed into inline data in operation codes in one or more of said instructions. Dependent claims 80, 98, and 126 also include this distinction. This is not disclosed in the cited references,

[13] Office Action p. 11.
[14] Wilkinson et al. at col. 10 lines 52-65. (emphasis added)

and is described in original claims 8, 21, and 35, and in the original specification at page 14 line 33 to page 15 line 1, page 17 line 21 to page 19 line 6, and FIGS. 5, 8A and 8B.

The Examiner contends the Wilkinson et al. reference discloses that some references to the constant pool are transformed into operation codes in instructions.[15] Again, contrary to the Examiner's contention, Wilkinson et al. does not disclose that some references to the constant pool are transformed into operation codes in instructions. Rather, the Wilkinson et al. reference discloses updating operands to byte codes that reference entries in a first constant pool to reflect their new location in a second constant pool.

In another aspect of the Applicant's claimed invention, an application software program comprises a sequence of instructions that were previously converted from at least one class file, and where the instructions comprise at least one composite instruction for performing an operation on a current object. This aspect is also completely missing from the cited references.

The Examiner contends the Wilkinson et al. reference discloses one composite instruction for performing an operation on a current object.[16] In support of the contention, the Examiner refers to Figure 7 of Wilkinson et al. The text accompanying FIG. 7 discloses:

> Typically, the translated byte codes are not interpreted in the Card JVM 16 but are supported by converting the byte codes into equivalent byte codes that can be interpreted by the Card JVM 16 (see FIG. 7). The byte codes 70 may be replaced with another semantically equivalent but different byte codes 72. This generally entails the translation of short single specific byte codes such as ILOAD_0 into their more general versions. For example, ILOAD_0 may be replaced by byte code ILOAD with an argument 0.[17]

[15] Office Action p. 11.
[16] Office Action p. 5.

The Applicants respectfully submit the Examiners attempt to equate a "generalized" version of a byte code with a composite instruction is improper. The "generalized" version of a byte code disclosed by Wilkinson et al. replaces byte codes with another semantically equivalent but different byte code, whereas the composite instruction as disclosed and claimed has no semantic equivalence requirement. Furthermore, the Applicants respectfully submit the Examiners' attempt to equate a value (0) corresponding to a specific byte code (ILOAD_0) in the above example with a "current object" is improper.

Independent claims 69, 87, 109, and 137 contain this distinction and are thus both novel and unobvious over the cited references. Independent claim 69 claims an object-oriented, verifiable, type-safe and pointer-safe sequence of instructions residing on a computer-readable medium, said instructions comprising operation codes and operands, said program operable to be loaded to and executed by a resource-constrained device, said instructions previously converted from at least one class file, said instructions comprising at least one composite instruction for performing an operation on a current object. Independent claim 87 claims a resource-constrained device configured to execute the application software program. Independent claim 109 claims a method for using the application software program. Independent claim 137 is a means-plus-function claim corresponding to claim 95.

Support for new claims 69, 87, 109, and 137 is provided in the original specification at page 6 line 31 to page 7 line 2, page 15 lines 2-4, page 20 lines 8-27, and FIGS. 10A and 10B. Support for new claims 69, 87, 109, and 137 is also provided in original claims 14, 37, and 53.

[17] Wilkinson et al. at col. 10 lines 35-44.

Dependent claims 70-76, 88-94, 110-122, and 138-150 depend from claims 69, 87, 109, and 137, respectively. The base claims being allowable, the dependent claims must also be allowable.

Additionally, the new dependent claims 70-76, 88-94, 110-122, and 138-150 further define the invention and distinguish the prior art.

Thus, the Applicants respectfully assert that new claims 59-150 are both novel and unobvious over the cited references. In view of the foregoing, it is respectfully asserted that the claims are now in condition for allowance.

Request for Allowance

It is believed that this Amendment places the above-identified patent application into condition for allowance. Early favorable consideration of this Amendment is earnestly solicited.

If, in the opinion of the Examiner, an interview would expedite the prosecution of this application, the Examiner is invited to call the undersigned attorney at the number indicated below. The Commissioner is hereby authorized to charge any additional fees or credit any overpayment to Deposit Account No. 50-1698.

Respectfully submitted,
THELEN REID & PRIEST, LLP

Dated: May 23, 2003

John P. Schaub
Reg. No. 42,125

THELEN REID & PRIEST LLP
P. O. Box 640640
San Jose, CA 95164-0640
Tel: (408) 292-5800